第 28 卷第 6 期　　中国科学院研究生院学报　　Vol. 28　No. 6
2011 年 11 月　　Journal of Graduate University of Chinese Academy of Sciences　　November　2011

文章编号:1002-1175(2011)06-0786-07

L4 进程间通信机制的模型检测方法*

高妍妍†,李　曦,周学海

(中国科学技术大学计算机科学与技术学院,合肥 230027;中国科学技术大学苏州研究院,苏州 215123)

(2010 年 7 月 19 日收稿;2010 年 11 月 9 日收修改稿)

Gao Y Y, Li X, Zhou X H. Verification of the L4 IPC implementation[J]. Journal of Graduate University of Chinese Academy of Sciences,2011,28(6):786-792.

摘　要　采用模型检测方法验证微内核操作系统的进程间通信机制,提出了一种从源码提取验证模型的方法.该方法以 L4 操作系统的进程间通信机制的C++源码实现为检验对象,从源码实现直接提取形式化模型,得到 Promela 语言的模型描述,可以直接应用模型检测器 Spin 对其进行正确性检测.实验表明了该方法的可行性和实用性.

关键词　L4 微内核,进程间通信机制,模型检测,Spin

中图分类号　TP302

随着计算机技术的发展,嵌入式系统已经广泛应用于社会生活的各个领域.微内核操作系统是许多嵌入式计算系统的核心组成部分,微内核的正确性是整个计算系统正确性的基础.传统的操作系统正确性检测采用测试的方法进行,其难点在于测试用例的设计和选取.在面对复杂系统时需要构建可以检查所有状态组合的实例集合,极大地增加了实现和应用上的难度,延长了测试时间,且难以发现全部问题.形式化方法利用数学手段描述和检查系统的行为,从而能够帮助研究人员发现其他方法不易发现的系统描述不一致、不明确或不完整等问题[1],因此,形式化方法被越来越多的研究人员引入到软硬件系统的验证工作中.作为数学方法,形式化方法在使用时需要验证人员提供待验证系统的数学模型,用于系统的验证工作.因此,如何有效建立待验证模型成为形式化方法应用中的一个关键问题.

形式化方法已经在一些项目中得到应用[2-4].在这些实例中,通常采用自上而下逐步细化的方法,在设计过程的开始阶段就伴随形式化的工作,通过对抽象描述建模和验证实现在设计早期的验证工作,从而保证各个设计阶段的正确性和一致性.还有一些工作[5-6]是通过分析源码和查阅说明文档建立抽象模型,实现对系统属性的验证.但是这些方法或者不直接涉及源码验证,或者需要研究人员在建立模型前对源码进行深入的分析以便有针对性地进行模型提取.但是,对于已经实现的系统,很多缺乏完备的流程文档,使得模型提取变得困难.但对于已有的大量的遗留代码,从源码开始的验证工作是很有必要的.本文的工作针对已有的 L4 源码展开.

作为一种主要的形式化验证方法,模型检测方法[7]自 1981 年由 Clarke 和 Emerson 提出以来,对该方法在软/硬件系统验证中的应用,以及验证方法本身的研究一直都在进行.目前已有多种模型检测工具,如 SMV、COSPAN/FORMAL CHECK、Spin 等.Spin 以 Promela 为输入语言,适合于并行系统.随着模型检测方法的发展,用模型检测方法验证操作系统的研究也开始增多.在这些工作中,多采用 Spin 作为模型检测工具.文献[8-10]分别用 Promela 描述了微内核操作系统 Harmony、RUBIS 和 Fluke 的进程间

* 国家"863"高技术研究发展计划项目(2008AA01Z101)资助

†E-mail:yygao@mail.ustc.edu.cn

通信机制,并用 Spin 模型检测器对它们进行验证,这些操作系统的结构较为简单.在文献[6]中 Endrawaty 建模和验证了 Fiasco 中的进程间通信机制. Fiasco 是针对 x86 结构的 L4 接口的一种实现.作者提供了从C ++ 源码建模的 Promela 模型,并对 IPC 的状态属性和路径属性进行检验.文献[11]是对 Linux 进程间通信进行模型检测的研究,作者分析 Linux 内核 2.6.0 版本中的相关源码,从中抽取有限状态自动机(FSA)描述的形式化模型,再将模型向 Promela 代码转换.在文献中建模了 Linux 中的管道通信和 socket 进程间通信机制,并对管道通信以及 socket 进程间通信机制中的可终止性和死锁等属性进行了检测.这些方法通过对相应操作系统的建模,帮助发现并发错误,但手动构建使其抽象不一定是健全的,因此不能保证正确性.本文所做工作致力于在形式化过程中减少人工参与部分,减少人为因素对正确性检测的影响.该方法以源码为建模对象,通过对源语言和目标语言的语法和句法结构的分析,实现源语言到目标语言的转化,建立形式化的模型描述.

1　Spin 与 Promela

Spin 适合于并行系统,尤其是协议一致性的辅助分析检测.包括 1996 年被 NASA 用于检查火星探测者所存在的错误在内,Spin 已经被广泛应用于工业界和学术界. Spin 以 Promela 为输入语言描述待检测的系统.用线性时序逻辑(linear temporal logic,LTL)描述待验证的属性,Spin 在验证时采用 on-the-fly 技术,从而可以根据需要生成系统自动机的部分状态,而无需构建全局的状态图或 Kripke 结构.此外,Spin 还支持对系统的模拟.通过由 Promela 描述生成一个 C 代码程序,实现对系统执行的模拟.

Promela(Protocol/Meta Language)是用来对有限状态系统进行建模的形式化描述语言.它类似于 C 语言,允许动态创建并行执行的进程,并可以在进程间通过消息通道进行同步.一个模型由类型说明、消息通道说明、变量说明和进程说明组成.

1)基本的数据类型. Promela 中基本的数据类型包括 bit、bool、byte、short 和 int.复杂数据类型包括数组、枚举类型(mtype)和结构定义.

2)消息通道.在 Promela 中,消息通道用于在进程之间进行消息传递,通道按先进先出顺序传递消息.

3)语句类型.语句类型除了一般语言中常见的赋值语句、条件语句、选择语句、重复语句(do)、无条件跳转语句(goto)、break 语句和断言(assert),还包含专门用于描述程序并行执行的语句:消息发送和接收语句、超时语句(timeout)和原子序列(atomic).

4)进程说明.进程的标识符为"proctype",它由进程名、参数列表、局部变量说明和进程体组成.进程体由一系列的语句组成.一个进程说明可以对应多个进程实例.进程实例可以在任何进程中使用 run 语句创建,也可以通过直接在进程前加上关键字 active 实现.

在 Promela 中,用关键字"init"标志的进程类似于 C 语言中的 main()函数.

在研究中,我们选择 Spin 作为模型检测方法的实现工具,因此用 Promela 作为验证模型的描述语言.在第 2 节,通过具体的转换规则说明如何实现从C ++ 源程序到 Promela 模型的转化.

2　进程间通信机制建模

针对进程间通信机制的执行特点,以及源语言和目标语言的语法和句法特征,确定具体的模型提取规则.

2.1　进程间通信机制

L4Ka::Pistachio[12]是 Karlsruhe 大学开发的最新的 L4 微内核,可运行在 AMD64、IA32、PowerPC32 位和 PowerPC64 位体系结构上.本文以 L4Ka::Pistachio 中进程间通信机制(inter-process communication,IPC)的实现为验证对象,对其进行正确性检验.

在 L4ka::Pistachio 中,进程可以利用 IPC 系统调用完成消息的发送和接收动作. IPC 是进程间通信和同步的基本操作.所有的通信是同步和无缓冲的:一条消息从发送者发送到接收者当且仅当这个接收

者已经调用了一个对应的 IPC 操作. 在发送过程中若接收者没有调用对应的 IPC 接收操作,发送者被阻塞,直到接收者调用了对应的 IPC 操作或发送者指定的时间已过.

L4ka::Pistachio 进程间通信机制的实现采用C ++ 编码,使用类定义数据对象. IPC 机制在实现线程间的消息传递时需要与内核中的另外 2 个基本结构——线程和地址空间——进行交互,因此执行代码涉及内核中大量不同的类实例. 主要的操作对象包括通信线程自身的线程控制块中的状态和数据信息、以及与线程相关的调度器信息等. 在 IPC 机制的执行函数中,除了对这些类中的数据的访问之外,还需要调用到类中的某些相关函数.

在形式化过程中将所有相关的类作为数据说明和内嵌函数的组合处理是最直接的方法. 但在程序实现中,每次对类的访问通常只涉及该类中的部分数据和函数. 此外,在模型检测方法中,过多的数据对象和函数易导致状态空间爆炸问题,因此在建模过程中我们进行了一些取舍,即对所有的 IPC 实现源码我们完全从C ++ 源码建立模型;而对于过程中涉及到的其他机制,我们将其作为 IPC 形式化模型的运行环境,仅对其进行抽象建模.

2.2 从C ++ 源码建模 IPC 机制

Promela 语言从 C 语言的一个子集扩展而来. C ++ 作为对象语言,比 C 语言的结构还要复杂和灵活,这使得从C ++ 源码直接提取 Promela 描述的模型变得困难. 在这一节,简要分析转换过程中的一些关键点.

2.2.1 模块化提取方法

函数由变量声明和语句序列组成. 在语句序列中可以包含函数调用. 因此,一个函数除了直接调用的子函数,还可能存在间接调用甚至是多层间接调用. 针对函数多层调用的情况,我们将每个函数作为一个独立的对象进行模型提取和语言转换. 在需要缩减规模时,用功能描述替换函数的具体执行过程,实现函数描述的简化和规避局部变量对状态空间的占用.

2.2.2 数据类型转换

数据类型声明和语句序列是程序的基本组成部分. 由上一节的分析可知,Promela 具有C ++ 语言包含的多数数据类型和语句类型,因此在这二者上易于建立直接的对应关系. 数据转换过程中的难点在于指针和C ++ 类的转换.

规则 1. 类提取. 将出现在C ++ 程序的数据声明部分中的每个类定义(CPPClass_x)转化为 Promela 语言中的"类"定义(PromelaClass_x_t)的方法,将 PromelaClass_x _t 声明为一个结构体,然后对 CPPClass _x 中的每个数据声明执行如下转换过程

1)将 CPPClass_x 定义中的基本数据类型和复杂数据类型说明转换为 Promela 中对应的数据类型,并将转换后的结果作为结构体中的数据说明;

2)对于 CPPClass_x 定义中出现的类指针说明 CPPClass_y * ,如未提取过类 CPPClass_y,则跳转到 1)提取该类. CPPClass_y 的实例个数与 CPPClass_x 相同.

规则 2. 类说明和类指针提取. 对C ++ 程序中每个类定义 CPPClass_x,执行规则 1 中的类提取后:

1)在 PromelaClass_x_t 声明后以数组形式声明 PromelaClass_x_t 的实例化个数,如需要提取 N 个 PromelaClass_x_t 实例的声明为

PromelaClass_x_t PromelaClass_x[N];

2)对C ++ 程序中使用到的类指针说明(CPPClass_x * p),用 PromelaClass_x_t 的实例个数的取值范围约束 p 的类型说明,如 N 的取值范围为 0 ~ 8,则 p 可以声明为 byte p;

3)对C ++ 程序中使用到类指针的位置(p- >),将指针替换为具体的类实例形式 PromelaClass_x [p].

在C ++ 程序中,类结构由多种类型的数据说明和一些对类中定义的数据进行操作的函数组成. 尽管这些数据和函数都被封装在一个类中,通过该类定义的数据对象访问,而在实际执行过程中每次对类的访问仅限于其中的部分数据和函数,因此,在向 Promela 描述转化时仍然将每个类作为一个整体处

理,但在转化后的类描述中仅保留在待形式化的源码中使用到的数据说明. 转化时,每个类被定义为一个数组类型,每个元素都是一个复合结构,这个结构由该类中声明的数据类型组成,包含相关源码中使用到的所有简单数据说明和复杂数据说明,数组的元素个数由该类在执行过程中所需的实例个数确定. 在这种情况下,被声明为指针的变量对应于数组的索引.

Promela 中没有指针的概念,在转化时需要依据具体情况定义转化方法,如在指向类时需要将其声明为非负数,并在使用时被替换为数组元素的形式.

2.2.3　语句转换

在 IPC 程序中,使用到的语句主要是赋值语句和选择语句. 赋值语句的表示形式与 Promela 中的类似. 对选择语句的转换,除了单纯的格式修改,还要注意语句含义的提取. 在C ++ 语法中,选择语句仅在条件满足时执行,如对于没有 else 辅助的 if 语句. 但是在 Promela 语法中,如果选择语句中的所有条件都不被满足,该语句被阻塞,直到其中一条语句的执行条件被满足. 因此,在这种情况下,需要在转换时添加一个执行空操作的 else 序列.

规则 3. 选择语句的提取.

1)对于不带有默认操作的选择语句(即在分支语句 if 中没有对应的 else 部分,或在多分支语句 switch-case 中没有 default 部分的情况),在提取时用空操作{}补足缺失部分(else 部分或 default 部分)的描述;

2)将选择语句转换为目标语言的句法描述,得到的每个语句序列表示一种可能的执行选择;

3)对于每个序列中的语句,根据句法规则执行句法转换,若遇到空操作{}将其转换为 skip 语句;若存在选择语句,则执行 1).

Promela 语言不支持 return 语句的使用,但在C ++ 语言中,return 语句被大量用于赋值函数,它不但用于赋值函数的返回值,还标志一个函数执行结束的返回点. 因此,在提取 return 语句时需要对整个函数进行调整.

规则 4. return 语句提取时的结构转换.

1)分析函数结构,将函数分解为选择语句组成的语句块和相邻选择语句间的语句序列组成的语句块;

2)在每个语句块中查找“return”标识符,如在具有 n 个序列的选择语句中的 x 个序列中出现了“return”标识符,则将该选择语句后的所有语句复制$(n-x-1)$份后分别移动到这$(n-x)$个序列的执行序列中;

3)对于修改后的模型,检查每个序列中的语句块,若该块不为空,则执行 1);否则结束.

对 return 作为返回值赋值的功能,在函数转换一节中进行提取.

2.2.4　函数转换

在 IPC 程序中调用了大量的函数,这些函数既有具体的功能实现,也有对类中定义的对类中元素执行简单操作的函数. 为了转换结果的简单直观,功能函数用内嵌函数方式定义和引用,类中的简单函数直接用具体操作替换,如对以“get_”和“set_”为前缀的类元素的取值和赋值操作,直接将函数表示为对应元素在 Promela 中的表示方式.

对于功能函数的转换,采用规则 1 中的模块化提取方法对每个函数进行独立处理. 在C ++ 语言中,函数大致可分为 2 类:执行一定功能的过程调用和带有返回值的函数. 在 Promela 语言中不支持用函数返回值,因此,对于带有返回值的函数需要执行额外的转换工作,这些工作用规则 5 和规则 6 描述.

规则 5. 对赋值函数的引用规则. 对于C ++ 中带有返回值的函数 type_proc_i proc_i (decl_list),其中 type_proc_i 为函数的返回值类型,

1)在引用语句前增加一个记录返回值的变量声明和函数调用描述:

```
type_proc_i return_proc_i;
proc_i(decl_list');
```

其中(decl_list')为(decl_list, return_proc_i).

2)将源程序中对函数"proc_i (decl_list)"的调用用返回值 return_proc_i 替换.

规则6. 对赋值函数的预处理. 对于C++中带有返回值的函数 type_proc_i proc_i (decl_list),其中 type_proc_i 为函数的返回值类型,

1)在类型声明 decl_list 中加入一个 type_proc_i 类型的参数 return_val 用于保存函数返回结果;

2)根据规则4调整函数结构;

3)将函数中的返回语句"return x;"替换为"return_val = x;".

以比较函数 max()为例,若源程序为

```
if  max(a, b)  {x = a;}
else {x = b;}
```

则依据提取规则将描述转换为:

```
  bool return_max;
  max(a, b, return_max);
if
:: return_max -> {x = a;}
:: else ->  {x = b;}
  fi;
```

在 Promela 中,C++函数在功能上对应于函数或进程. 在进程间通信机制的模型提取中,我们将进程通信机制的主功能函数作为进程处理,将该函数引用的过程和函数看作内嵌函数(inline).

规则7. 内嵌函数的提取. 一个函数 Proc 的转换过程:

1)若函数为赋值函数,执行规则6中的预处理操作;

2)若函数带有参数列表,去掉参数列表中的类型说明部分,转化为目标语言中的参数列表形式;

3)依据数据提取规则将变量声明 var_decl 转换为目标语言描述并将其声明为所属进程的局部变量;

4)依据句法转换规则将 Proc 中的语句序列转换为目标语言的句法;

5)对 Proc 中存在的函数调用 Proc_i,若 Proc_i 为赋值函数,用规则5对函数引用进行转换,否则直接进行如下转换:

i)若函数 Proc 支持对深层函数的调用,保留对函数 Proc_i 的调用,检查函数 Proc_i 是否被定义,若没有被定义则跳转到1)执行对函数 Proc_i 的提取;

ii)若函数 Proc 不支持深层函数调用,用与函数 Proc_i 功能等价的语句序列替换函数引用.

由于函数可能在调用者函数中被调用多次,导致局部变量在调用者函数中被重复声明,因此,在函数转换过程中要对函数的局部变量声明进行修改.

3 实验及结果分析

以 L4Ka::Pistachio 中的 IPC 源码在单处理器上的执行源码为例,用第2节的方法对单处理器上的 IPC 函数 SYS_IPC 进行转化,得到对应的 Promela 程序. 该程序可直接作为模型检测器的输入实现语义检查.

为了避免程序的验证受实验环境的局限,我们对模型进行了简化:仅考虑 IPC 源码中直接调用的函数,即在这些被调用函数中不支持对它的子函数的调用,将子函数用对应的功能描述替换,从而避免子函数中的局部变量占用过多空间,使程序验证可在普通 PC 机上进行:

1)对于以"get_"和"set_"为前缀的针对类数据对象的操作(这里简记为 get_*()和 set_*()),直接用赋值语句描述;

2)对轨迹记录函数 TRACE_IPC_DETAILS 和 TRACE_IPC_ERROR,用 printf()语句描述,将该次引

用作为 printf( )语句的参数;

3)对返回值函数 retrun_ipc( ),对于其中的退出功能描述部分,用跳转语句 goto 实现,在 SYS_IPC 程序的结尾部分添加一个标志 SYS_IPC_Label 作为辅助,用 goto SYS_IPC_Label 实现 retrun_ipc( )函数中的退出功能;

4)对队列操作 set_timeout( )、cancel_timeout( )、enqueue_snd( )、dequeue_snd( )和 enqueue_ready( ),由于在模型中仅考虑 SYS_IPC 直接调用的函数,对这些队列函数中调用的队列修改函数,用功能模块描述;

5)对优先级比较函数 check_dispatch_thread( ),为了缩减空间规模,用随机选择函数替换;

6)由于在单 CPU 情形中仅设置锁但并未将锁作为调度条件,在提取后的模型中省略了锁函数 lock( )和 unlock( );

7)对消息传递函数 transfer_message( ),在模型检测阶段,用功能描述替换具体的执行语句.

在执行模型检测时,还要设置程序执行的环境,即对内核中 IPC 机制以外的部分进行抽象.对环境的设置不是本文关注的内容,为了检验转换结果的正确性,这里仅进行一些简单的设置:进程间通信在2个线程 sender_thread 和 receiver_thread 间进行;程序中使用的所有数据变量的初值在这些变量在程序初始时刻可取值范围内随机生成;在线程切换函数(switch_to)中,通过添加相应的语句序列,在其中附加了调度器功能,实现发送进程和接收进程间的切换.

采用以上配置对 IPC 源码转换得到的 Promela 程序进行了验证. Spin 搜索了全部的 1445473 次状态迁移,程序中设置的全部断言均被满足.

4　结束语

在软件程序和操作系统代码的正确性验证过程中,如何提取程序模型是一个关键问题.对于已经开发出的程序,特别是对于大量的遗留代码,需要从程序源码直接提取验证模型.在本文中,针对 L4Ka::Pistachio 中的进程间通信机制的验证,提出直接从C++源码建模进程间通信机制的方法,通过在2种语言的数据对象和句法规则间建立对应关系,实现对数据声明、语句序列和调用函数的转换.方法本身具有普适性,在程序简单修改后可以通过修改模型中的对应部分避免重新建模,且过程的大部分易于自动化实现.实验表明,转换过程是可行的.下一步工作将研究如何构建支持 SMP 的进程间通信机制模型,及研究如何构建模型的运行环境和属性集合.

参考文献

[1] Wing J M. A symbiotic relationship between formal methods and security[C]// Workshops on Computer Security, Dependability, and Assurance: From Needs to Solutions. Washington, DC, USA: IEEE Computer Society, 1998.

[2] Klein G. Operating system verification—an overview[J]. Sādhanā, 2009, 34(1): 26-69.

[3] Cock D, Klein G, Sewell T. Secure microkernels, state monads and scalable refinement[C]// Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics. Berlin, Heidelberg: Springer-Verlag, 2008.

[4] Hillebrand M A, Paul W J. On the architecture of system verification environments[C]// Proceedings of the 3rd International Haifa Verification Conference on Hardware and Software: Verification and Testing. Lecture Notes in Computer Science, 4899. Berlin, Germany: Springer-Verlag, 2008.

[5] Hohmuth M, Tews H, Stephens S G. Applying source-code verification to a microkernel: the VFiasco project[C]// Proceedings of the 10th ACM SIGOPS European Workshop. New York, NY, USA: ACM, 2002.

[6] Endrawaty E. Verification of the fiasco IPC implementation[D]. Department of Computer Science, Institute of Theoretical Computer Science. Dresden University of Technology, Germany. 2005.

[7] Clarke E, Grumberg O, Long D. Verification tools for finite-state concurrent systems[C]// A Decade of Concurrency-Reflections and Perspectives. Lecture Notes in Computer Science, 803. London, UK: Springer-Verlag, 1994: 124-175.

[8] Cattel T. Modelization and verification of a multiprocessor realtime OS kernel[C]// Proceedings of FORTE'94. London, UK: Chapman & Hall, Ltd, 1995:55-70.

[9] Duval G, Julliand J. Modelling and verification of the RUBIS μ-kernel with SPIN[C]// SPIN'95. Workshop on Model Checking of Software, 1995.

[10] Tullmann P, Turner J, McCorquodale J, et al. Formal methods: a practical tool for OS implementors[C]// HotOS-VI. Washington, DC, USA: IEEE Computer Society, 1997.

[11] 姜玉蓉. LINUX 内核进程间通信的模型检测研究[D]. 长沙: 湖南大学, 2009.

[12] Universität Karlsruhe. L4Ka::Pistachio microkernel[EB/OL]. [2010-05-02]. http://l4ka.org/projects/pistachio/.

Verification of the L4 IPC implementation

GAO Yan-Yan, LI Xi, ZHOU Xue-Hai

(*Department of Computer Science and Technology, University of Science and Technology of China, Hefei* 230027, *China*;
Embedded System Laboratory, Suzhou Institute for Advanced Study, USTC, Suzhou 215123, *China*)

Abstract Inter-process communication (IPC) mechanism is one of the key technologies of microkernel operating system. In this paper we present a formal method to model and verify the IPC implementation. The source code of L4 IPC implementation is translated into abstract model which is described in Promela, and the abstract model can be verified with Spin directly. The experimental results show the feasibility and practicality of the method.

Key words L4 microkernel, inter-process communication(IPC), model checking, Spin

L4进程间通信机制的模型检测方法

作者： 高妍妍， 李曦， 周学海， GAO Yan-Yan， LI Xi， ZHOU Xue-Hai
作者单位： 中国科学技术大学计算机科学与技术学院,合肥230027;中国科学技术大学苏州研究院,苏州215123
刊名： 中国科学院研究生院学报ISTIC PKU
英文刊名： Journal of the Graduate School of the Chinese Academy of Sciences
年，卷(期)： 2011,28(6)

参考文献(12条)

1. Wing J M A symbiotic relationship between formal methods and security 1998
2. Klein G Operating system verification-an overview[外文期刊] 2009(01)
3. Cock D;Klein G;Sewell T Secure microkernels,state monads and scalable refinement 2008
4. Hillebrand M A;Paul W J On the architecture of system verification environments[外文期刊] 2008
5. Hohmuth M;Tews H;Stephens S G Applying source-code verification to a microkernel:the VFiaaco project 2002
6. Endrawaty E Verification of the fiasco IPC implementation 2005
7. Clarke E;Grumberg O;Long D Verification tools for finite-state concurrent systems 1994
8. Cattel T Modelization and verification of a multiprocessor realtime OS kernel 1995
9. Duval G;Julliand J Modelling and verification of the RUBIS μ-kernel with SPIN 1995
10. Tullmann P;Turner J;McCorquodale J Formal methods ; a practical tool for OS implementors 1997
11. 姜玉蓉 LINUX内核进程间通信的模型检测研究[学位论文] 2009
12. Universit(a)t Karlsruhe L4Ka::Pistachio microkernel 2010